

- *Regularization tuning parameters.* These include the dropout rate ϕ and the strength λ of lasso and ridge regularization, and are typically set separately at each layer.
- *Details of stochastic gradient descent.* These includes the batch size, the number of epochs, and if used, details of data augmentation (Section 10.3.4.)

Choices such as these can make a difference. In preparing this **MNIST** example, we achieved a respectable 1.8% misclassification error after some trial and error. Finer tuning and training of a similar network can get under 1% error on these data, but the tinkering process can be tedious, and can result in overfitting if done carelessly.

10.8 Interpolation and Double Descent

Throughout this book, we have repeatedly discussed the bias-variance trade-off, first presented in Section 2.2.2. This trade-off indicates that statistical learning methods tend to perform the best, in terms of test-set error, for an intermediate level of model complexity. In particular, if we plot “flexibility” on the x -axis and error on the y -axis, then we generally expect to see that test error has a U-shape, whereas training error decreases monotonically. Two “typical” examples of this behavior can be seen in the right-hand panel of Figure 2.9 on page 31, and in Figure 2.17 on page 42. One implication of the bias-variance trade-off is that it is generally not a good idea to *interpolate* the training data — that is, to get zero training error — since that will often result in very high test error.

However, it turns out that in certain specific settings it can be possible for a statistical learning method that interpolates the training data to perform well — or at least, better than a slightly less complex model that does not quite interpolate the data. This phenomenon is known as *double descent*, and is displayed in Figure 10.20. “Double descent” gets its name from the fact that the test error has a U-shape before the interpolation threshold is reached, and then it descends again (for a while, at least) as an increasingly flexible model is fit.

We now describe the set-up that resulted in Figure 10.20. We simulated $n = 20$ observations from the model

$$Y = \sin(X) + \epsilon,$$

where $X \sim U[-5, 5]$ (uniform distribution), and $\epsilon \sim N(0, \sigma^2)$ with $\sigma = 0.3$. We then fit a natural spline to the data, as described in Section 7.4, with d degrees of freedom.²³ Recall from Section 7.4 that fitting a natural spline

²³This implies the choice of d knots, here chosen at d equi-probability quantiles of the training data. When $d > n$, the quantiles are found by interpolation.

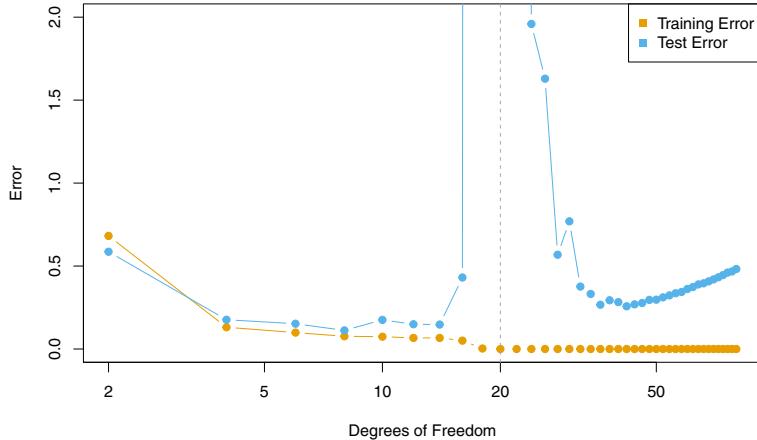


FIGURE 10.20. Double descent phenomenon, illustrated using error plots for a one-dimensional natural spline example. The horizontal axis refers to the number of spline basis functions on the log scale. The training error hits zero when the degrees of freedom coincides with the sample size $n = 20$, the “interpolation threshold”, and remains zero thereafter. The test error increases dramatically at this threshold, but then descends again to a reasonable value before finally increasing again.

with d degrees of freedom amounts to fitting a least-squares regression of the response onto a set of d basis functions. The upper-left panel of Figure 10.21 shows the data, the true function $f(X)$, and $\hat{f}_8(X)$, the fitted natural spline with $d = 8$ degrees of freedom.

Next, we fit a natural spline with $d = 20$ degrees of freedom. Since $n = 20$, this means that $n = d$, and we have zero training error; in other words, we have interpolated the training data! We can see from the top-right panel of Figure 10.21 that $\hat{f}_{20}(X)$ makes wild excursions, and hence the test error will be large.

We now continue to fit natural splines to the data, with increasing values of d . For $d > 20$, the least squares regression of Y onto d basis functions is not unique: there are an infinite number of least squares coefficient estimates that achieve zero error. To select among them, we choose the one with the smallest sum of squared coefficients, $\sum_{j=1}^d \hat{\beta}_j^2$. This is known as the *minimum-norm* solution.

The two lower panels of Figure 10.21 show the minimum-norm natural spline fits with $d = 42$ and $d = 80$ degrees of freedom. Incredibly, $\hat{f}_{42}(X)$ is quite a bit less wild than $\hat{f}_{20}(X)$, even though it makes use of more degrees of freedom. And $\hat{f}_{80}(X)$ is not much different. How can this be? Essentially, $\hat{f}_{20}(X)$ is very wild because there is just a single way to interpolate $n = 20$ observations using $d = 20$ basis functions, and that single way results in a somewhat extreme fitted function. By contrast, there are an

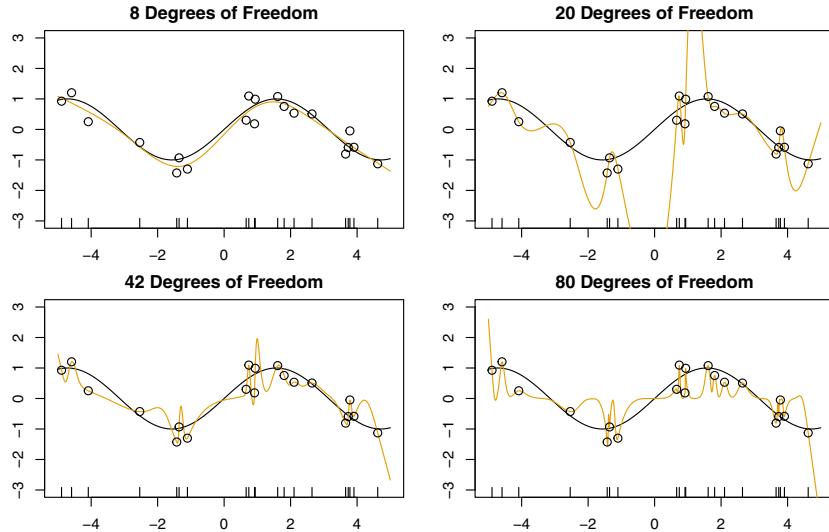


FIGURE 10.21. Fitted functions $\hat{f}_d(X)$ (orange), true function $f(X)$ (black) and the observed 20 training data points. A different value of d (degrees of freedom) is used in each panel. For $d \geq 20$ the orange curves all interpolate the training points, and hence the training error is zero.

infinite number of ways to interpolate $n = 20$ observations using $d = 42$ or $d = 80$ basis functions, and the smoothest of them — that is, the minimum norm solution — is much less wild than $\hat{f}_{20}(X)$!

In Figure 10.20, we display the training error and test error associated with $\hat{f}_d(X)$, for a range of values of the degrees of freedom d . We see that the training error drops to zero once $d = 20$ and beyond; i.e. once the interpolation threshold is reached. By contrast, the test error shows a U -shape for $d \leq 20$, grows extremely large around $d = 20$, and then shows a second region of descent for $d > 20$. For this example the signal-to-noise ratio — $\text{Var}(f(X))/\sigma^2$ — is 5.9, which is quite high (the data points are close to the true curve). So an estimate that interpolates the data and does not wander too far inbetween the observed data points will likely do well.

In Figures 10.20 and 10.21, we have illustrated the double descent phenomenon in a simple one-dimensional setting using natural splines. However, it turns out that the same phenomenon can arise for deep learning. Basically, when we fit neural networks with a huge number of parameters, we are sometimes able to get good results with zero training error. This is particularly true in problems with high signal-to-noise ratio, such as natural image recognition and language translation, for example. This is because the techniques used to fit neural networks, including stochastic gradient descent, naturally lend themselves to selecting a “smooth” interpolating model that has good test-set performance on these kinds of problems.

Some points are worth emphasizing:

- *The double-descent phenomenon does not contradict the bias-variance trade-off, as presented in Section 2.2.2.* Rather, the double-descent curve seen in the right-hand side of Figure 10.20 is a consequence of the fact that the x -axis displays the number of spline basis functions used, which does not properly capture the true “flexibility” of models that interpolate the training data. Stated another way, in this example, the minimum-norm natural spline with $d = 42$ has lower variance than the natural spline with $d = 20$.
- *Most of the statistical learning methods seen in this book do not exhibit double descent.* For instance, regularization approaches typically do not interpolate the training data, and thus double descent does not occur. This is not a drawback of regularized methods: they can give great results *without interpolating the data!*

In particular, in the examples here, if we had fit the natural splines using ridge regression with an appropriately-chosen penalty rather than least squares, then we would not have seen double descent, and in fact would have obtained better test error results.

- *In Chapter 9, we saw that maximal margin classifiers and SVMs that have zero training error nonetheless often achieve very good test error.* This is in part because those methods seek smooth minimum norm solutions. This is similar to the fact that the minimum-norm natural spline can give good results with zero training error.
- *The double-descent phenomenon has been used by the machine learning community to explain the successful practice of using an over-parametrized neural network (many layers, and many hidden units), and then fitting all the way to zero training error.* However, fitting to zero error is not always optimal, and whether it is advisable depends on the signal-to-noise ratio. For instance, we may use ridge regularization to avoid overfitting a neural network, as in (10.31). In this case, provided that we use an appropriate choice for the tuning parameter λ , we will never interpolate the training data, and thus will not see the double descent phenomenon. Nonetheless we can get very good test-set performance, likely much better than we would have achieved had we interpolated the training data. Early stopping during stochastic gradient descent can also serve as a form of regularization that prevents us from interpolating the training data, while still getting very good results on test data.

To summarize: though double descent can sometimes occur in neural networks, we typically do not want to rely on this behavior. Moreover, it is important to remember that the bias-variance trade-off always holds (though

it is possible that test error as a function of flexibility may not exhibit a U-shape, depending on how we have parametrized the notion of “flexibility” on the x -axis).

10.9 Lab: Deep Learning

In this section, we show how to fit the examples discussed in the text. We use the `keras` package, which interfaces to the `tensorflow` package which in turn links to efficient `python` code. This code is impressively fast, and the package is well-structured. A good companion is the text *Deep Learning with R*²⁴, and most of our code is adapted from there.

Getting `keras` up and running on your computer can be a challenge. The book website www.statlearning.com gives step-by-step instructions on how to achieve this.²⁵ Guidance can also be found at `keras.rstudio.com`.

10.9.1 A Single Layer Network on the Hitters Data

We start by fitting the models in Section 10.6. We set up the data, and separate out a training and test set.

```
> library(ISLR2)
> Gitters <- na.omit(Hitters)
> n <- nrow(Gitters)
> set.seed(13)
> ntest <- trunc(n / 3)
> testid <- sample(1:n, ntest)
```

The linear model should be familiar, but we present it anyway.

```
> lfit <- lm(Salary ~ ., data = Gitters[-testid, ])
> lpred <- predict(lfit, Gitters[testid, ])
> with(Gitters[testid, ], mean(abs(lpred - Salary)))
[1] 254.6687
```

Notice the use of the `with()` command: the first argument is a dataframe, and the second an expression that can refer to elements of the dataframe by name. In this instance the dataframe corresponds to the test data and the expression computes the mean absolute prediction error on this data.

Next we fit the lasso using `glmnet`. Since this package does not use formulas, we create `x` and `y` first.

```
> x <- scale(model.matrix(Salary ~ . - 1, data = Gitters))
> y <- Gitters$Salary
```

²⁴F. Chollet and J.J. Allaire, *Deep Learning with R* (2018), Manning Publications.

²⁵Many thanks to Balasubramanian Narasimhan for preparing the `keras` installation instructions.